Computational Complexity Theory

Lecture 15: Class NC and AC; P-completeness

Department of Computer Science,
Indian Institute of Science

Recap: Lesson learnt from Cook-Levin

- Locality of computation implies that an algorithm A working on inputs of some fixed length n and running in time $T(n)$ can be viewed as a Boolean circuit ϕ of size $O(T(n)^2)$ s.t. $A(x) = \phi(x)$ for every $x \in \{0,1\}^n$.
- On the other hand, a circuit on inputs of length n and of size S can be viewed as an algorithm working on length n inputs and running in time S.

- To rule the existence of a sequence of algorithms – one for each input length – we need to rule out the existence of a sequence of <u>(i.e., a family of) circuits</u>.

Recap: Boolean circuits

- A <u>Boolean circuit</u> is a directed acyclic graph whose nodes/gates are labelled as follows:

  - A node with in-degree zero is labelled by an input variable, and it outputs the value of the variable.
  - Any other node is labelled by one of the three operations $\wedge$, $\vee$, $\neg$, and it outputs the value of the operation on its input.

  Nodes with out-degree zero are the output gates.

- <u>**Size**</u> of circuit is the no. of edges in it. <u>**Depth**</u> is the length of the longest path from an i/p to o/p node.

Recap: Class P/poly

- Let $T: \mathbb{N} \rightarrow \mathbb{N}$ be some function.
- Definition: A $T(n)$-size circuit family is a set of circuits $\{C_n\}_{n \in \mathbb{N}}$ such that C_n has n inputs and $|C_n| \leq T(n)$.

- Definition: A language L is in $SIZE(T(n))$ if there's a $T(n)$-size circuit family $\{C_n\}_{n \in \mathbb{N}}$ such that

$$x \in L \Leftrightarrow C_n(x) = 1, \text{ where } n = |x|.$$

- Defintion: Class $P/poly = \bigcup_{c \geq 1} SIZE(n^c)$.

Recap: Class P/poly

- Observation: $P \subseteq P/poly$.
- Proof. If $L \in P$, then there's a n^c-time TM that decides L for some constant c. By Cook-Levin, there's a $O(n^{2c})$-size circuit family $\{C_n\}_{n\in N}$ such that

$$x \in L \iff C_n(x) = 1, \text{ where } n = |x|.$$

(Note: C_n is poly(n)-time computable from 1^n.)

- Is $P = P/poly$? No! $P/poly$ contains undecidable languages.

Recap: Karp-Lipton theorem

- Theorem (*Karp & Lipton 1982*). If $\mathsf{NP} \subsetneq \mathsf{P/poly}$ then $\mathsf{PH} = \Sigma_2$.

- If we can show $\mathsf{NP} \not\subset \mathsf{P/poly}$ assuming $\mathsf{P} \neq \mathsf{NP}$, then

  $$\mathsf{NP} \not\subset \mathsf{P/poly} \Leftrightarrow \mathsf{P} \neq \mathsf{NP}.$$

- Karp-Lipton theorem shows $\mathsf{NP} \not\subset \mathsf{P/poly}$ assuming the stronger statement $\mathsf{PH} \neq \Sigma_2$.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly?

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many. Let $\exp(m) = 2^m$.
- Theorem. $1 - \exp(-2^{n-1})$ fraction of Boolean functions on n variables **do not** have circuits of size $2^n/(22n)$.
- Proof. Follows from a counting argument.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many. Let $\exp(m) = 2^m$.
- Theorem. $1 - \exp(-2^{n-1})$ fraction of Boolean functions on n variables **do not** have circuits of size $2^n/(22n)$.
- Proof. Let $s = 2^n/(22n)$. A circuit of size s has at most s internal nodes. It can be specified by giving the labels of the internal nodes and the adjacency lists.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many. Let $\exp(m) = 2^m$.
- Theorem. $1 - \exp(-2^{n-1})$ fraction of Boolean functions on n variables **do not** have circuits of size $2^n/(22n)$.
- Proof. Let $s = 2^n/(22n)$. A circuit of size s has at most s internal nodes. It can be specified by giving the labels of the internal nodes and the adjacency lists.
- Number of bits required to write the adjacency lists it at most $s(\log s + 3) + 4(s + n) \leq 9s.\log s$.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many. Let $\exp(m) = 2^m$.
- Theorem. $1 - \exp(-2^{n-1})$ fraction of Boolean functions on n variables **do not** have circuits of size $2^n/(22n)$.
- Proof. Let $s = 2^n/(22n)$. A circuit of size s has at most s internal nodes. It can be specified by giving the labels of the internal nodes and the adjacency lists.
- Number of circuits of size s is at most $3^s.2^{9s.\log s}$.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many. Let $\exp(m) = 2^m$.
- Theorem. $1 - \exp(-2^{n-1})$ fraction of Boolean functions on n variables **do not** have circuits of size $2^n/(22n)$.
- Proof. Let $s = 2^n/(22n)$. A circuit of size s has at most s internal nodes. It can be specified by giving the labels of the internal nodes and the adjacency lists.
- Number of circuits of size s is at most $2^{11 s.\log s}$.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many. Let $\exp(m) = 2^m$.
- Theorem. $1 - \exp(-2^{n-1})$ fraction of Boolean functions on n variables **do not** have circuits of size $2^n/(22n)$.
- Proof. Let $s = 2^n/(22n)$. A circuit of size s has at most s internal nodes. It can be specified by giving the labels of the internal nodes and the adjacency lists.
- Number of circuits of size s is at most $\exp(2^{n-1})$.
- Number of functions in n variables is $\exp(2^n)$.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many. Let $\exp(m) = 2^m$.
- Theorem. $1 - \exp(-2^{n-1})$ fraction of Boolean functions on n variables **do not** have circuits of size $2^n/(22n)$.
- Proof. Let $s = 2^n/(22n)$. A circuit of size s has at most s internal nodes. It can be specified by giving the labels of the internal nodes and the adjacency lists.
- So, circuits of size s can compute at most $\exp(-2^{n-1})$ fraction of all Boolean functions on n variables.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many.
- Is one out of so many functions outside P/poly in NP?

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many.
- Is one out of so many functions outside P/poly in NP? We don't know even after ~40 yrs of research!

- Theorem. *(Iwama, Lachish, Morizumi & Raz 2002)* There is a language $L \in NP$ such that any circuit C_n that decides $L \cap \{0,1\}^n$ requires $5n - o(n)$ many $\wedge$ and $\vee$ gates.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many.
- Is one out of so many functions outside P/poly in NP? We don't know even after ~40 yrs of research!

- Theorem. *(Iwama, Lachish, Morizumi & Raz 2002)* There is a language $L \in NP$ such that any circuit C_n that decides $L \cap \{0,1\}^n$ requires $5n - o(n)$ many $\wedge$ and $\vee$ gates.

Results of this kind are known as circuit lower bound.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many.
- Is one out of so many functions outside P/poly in NP? We don't know even after ~40 yrs of research!

- Open problem. Prove that NEXP $\not\subset$ P/poly .

Lower bounds for restricted circuits

- Nevertheless, the clean combinatorial structure of a circuit has been used to prove lower bounds for some natural classes of circuits.
- The proofs of these lower bounds introduced and developed some highly interesting techniques.

Lower bounds for restricted circuits

- Nevertheless, the <u>clean combinatorial structure</u> of a circuit has been used to prove lower bounds for some <u>natural classes of circuits</u>.
- The proofs of these lower bounds introduced and developed some highly <u>interesting techniques</u>.

- Fact. PARITY($x_1, x_2, \ldots, x_n$) can be computed by a circuit of size $O(n)$ and a formula of size $O(n^2)$.

Homework

Lower bound for Boolean formulas

- Nevertheless, the clean combinatorial structure of a circuit has been used to prove lower bounds for some natural classes of circuits.
- The proofs of these lower bounds introduced and developed some highly interesting techniques.

- Theorem. *(Khrapchenko 1971)* Any formula computing PARITY($x_1, x_2, \ldots, x_n$) has size $\Omega(n^2)$.

Lower bound for Boolean formulas

- Nevertheless, the clean combinatorial structure of a circuit has been used to prove lower bounds for some natural classes of circuits.
- The proofs of these lower bounds introduced and developed some highly interesting techniques.

- Theorem. (*Andreev 1987, Hastad 1998*) There's a f that can be computed by a $O(n)$-size circuit such that any formula computing f has size $\Omega(n^{3-o(1)})$.

Technique: Shrinkage of formulas under random restrictions (*Subbotovskaya 1961*).

Lower bound for Boolean formulas

- Nevertheless, the clean combinatorial structure of a circuit has been used to prove lower bounds for some natural classes of circuits.
- The proofs of these lower bounds introduced and developed some highly interesting techniques.

- Conjecture. *(Circuits more powerful than formulas)* There's a f that can be computed by a $O(n)$-size circuit such that any formula computing f has size $n^{\omega(1)}$.

An interesting approach was given by *Karchmer, Raz & Wigderson (1995)* .

LB for AC^0 and monotone circuits

- Nevertheless, the clean combinatorial structure of a circuit has been used to prove lower bounds for some natural classes of circuits.
- The proofs of these lower bounds introduced and developed some highly interesting techniques.

- We'll discuss a **super-polynomial** lower bound for constant depth circuits later.

Non-uniform size hierarchy

- Shanon's result. There's a constant $c \geq 1$ such that every Boolean function in n variables has a circuit of size at most $c.(2^n/n)$.

- Theorem. There's a constant $d \geq 1$ s.t. if $T_1: \mathbb{N} \to \mathbb{N}$ & $T_2: \mathbb{N} \to \mathbb{N}$ and $T_1(n) \leq d^{-1}.T_2(n) \leq T_2(n) \leq c.(2^n/n)$ then

$$\mathrm{SIZE}(T_1(n)) \subsetneq \mathrm{SIZE}(T_2(n)).$$

Non-uniform size hierarchy

- Shanon's result. There's a constant $c \geq 1$ such that every Boolean function in n variables has a circuit of size at most $c.(2^n/n)$.

- Theorem. There's a constant $d \geq 1$ s.t. if $T_1: \mathbb{N} \rightarrow \mathbb{N}$ & $T_2: \mathbb{N} \rightarrow \mathbb{N}$ and $T_1(n) \leq d^{-1}.T_2(n) \leq T_2(n) \leq c.(2^n/n)$ then

$$\mathrm{SIZE}(T_1(n)) \subsetneq \mathrm{SIZE}(T_2(n)).$$

- Proof. Uses Shanon's result and a counting argument.

(*Homework*)

Class NC^i and AC^i

Class NC

- NC stands for Nick's Class – named after Nick Pippenger.
- Definition. For $i \in \mathbb{N}$, a language L is in NC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.

- Definition. $NC = \bigcup_{i \in \mathbb{N}} NC^i$.

Class NC

- NC stands for Nick's Class – named after Nick Pippenger.
- Definition. For $i \in \mathbb{N}$, a language L is in NC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.
- Definition. $NC = \bigcup_{i \in \mathbb{N}} NC^i$.
- *Homework:* PARITY is in NC^1.

Class NC

- NC stands for Nick's Class – named after Nick Pippenger.
- Definition. For $i \in \mathbb{N}$, a language L is in NC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.

- Definition. $NC = \bigcup_{i \in \mathbb{N}} NC^i$.

- $\boxed{NC^1 = \textit{poly(n)}\text{-size Boolean formulas.}}$ (*Assignment*)

Class NC

- NC stands for Nick's Class – named after Nick Pippenger.
- Definition. For $i \in \mathbb{N}$, a language L is in NC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.

- Further, L is in ***log-space uniform*** NC^i if C_n is implicitly log-space computable from 1^n.

Note: Sometimes NC^i is defined as log-space uniform NC^i .

Class NC

- NC stands for Nick's Class – named after Nick Pippenger.
- Definition. For $i \in \mathbb{N}$, a language L is in NC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.

- Further, L is in ***log-space uniform*** NC^i if C_n is implicitly log-space computable from 1^n.

log-space uniform $NC \subseteq P$.

NC ≡ Efficient parallel computation

- Definition. A language L can be decided *efficiently in parallel* if there's a polynomial function $q(.)$ and constants c & i s.t. $L \cap \{0,1\}^n$ can be decided using $q(n)$ many processors in $c.(\log n)^i$ time.

NC ≡ Efficient parallel computation

- Definition. A language L can be decided *efficiently in parallel* if there's a polynomial function $q(.)$ and constants c & i s.t. $L \cap \{0,1\}^n$ can be decided using $q(n)$ many processors in $c.(\log n)^i$ time.
- Model: **PRAM** (has a central shared memory)
  - A processor can "deliver" a message to any other processor in $O(\log n)$ time.
  - A processor has $O(\log n)$ bits of memory and performs a poly-time computation at every step.
  - Processor computation steps are synchronized.

NC ≡ Efficient parallel computation

- Definition. A language L can be decided *efficiently in parallel* if there's a polynomial function $q(.)$ and constants c & i s.t. $L \cap \{0,1\}^n$ can be decided using $q(n)$ many processors in $c.(\log n)^i$ time.

- Observation. A language L is in NC if and only if L can be decided efficiently in parallel.
- Proof. Almost immediate from the assumptions on the parallel computation model.

Class AC

- Definition. For $i \in \mathbb{N} \cup \{0\}$, a language L is in AC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size **unbounded fan-in** circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.

- Definition. $AC = \bigcup_{i \geq 0} AC^i$. (stands for *Alternating Class*)

Class AC

- Definition. For $i \in \mathbb{N} \cup \{0\}$, a language L is in AC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size **unbounded fan-in** circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.

- Definition. $AC = \bigcup_{i \geq 0} AC^i$.

- Observation. $AC^i \subseteq NC^{i+1} \subseteq AC^{i+1}$ for all $i \geq 0$.

Replace an unbounded fan-in gate by a binary tree of bounded fan-in gates.

Class AC

- Definition. For $i \in \mathbb{N} \cup \{0\}$, a language L is in AC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size **unbounded fan-in** circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.

- Definition. $AC = \bigcup_{i \geq 0} AC^i$.

- Observation. $NC = AC$.

Class AC

- Definition. For $i \in \mathbb{N} \cup \{0\}$, a language L is in AC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size **unbounded fan-in** circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.

- Definition. $AC = \bigcup_{i \geq 0} AC^i$.

- In the next lecture, we'll show that PARITY is not in AC^0, i.e., $AC^0 \subsetneq NC^1$.

Class AC

- Definition. For $i \in \mathbb{N} \cup \{0\}$, a language L is in AC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size **unbounded fan-in** circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.

- Definition. $AC = \bigcup_{i \geq 0} AC^i$.

- Further, L is in ***log-space uniform*** AC^i if C_n is implicitly log-space computable from 1^n.

log-space uniform $AC \subseteq P$.

P-completeness

P-completeness

- Recall, to define completeness of a complexity class, we need an appropriate notion of a *reduction*.
- What kind of reductions will be suitable is guided by *a complexity question*, like a comparison between the complexity class under consideration & another class.
- Is P = (uniform) NC? Is P = L?…use log-space reduction!

- Definition. A language L ∈ P is *P-complete* if for every L' in P , L' $\leq_l$ L.

P-complete problems

- Circuit value problem. Given a circuit and an input, compute the output of the circuit. (The reduction in the Cook-Levin theorem can be made a log-space reduction.)

- Linear programming. Check the feasibility of a system of linear inequality constraints over rationals. (Assignment problem)

- CFG membership. Given a context-free grammar and a string, decide if the string can be generated by the grammar.

No log-space algo for PC problems

- Theorem. Let L be a P-complete language. Then,

  L is in L ⟺ P = L.

- Proof. Easy.

- Can't hope to get a log-space algorithm for a P-complete problem unless P = L.

No parallel algo for PC problems

- Theorem. Let L be a P-complete language. Then,

  L is in NC $\Leftrightarrow$ $P \subseteq NC$.

- Proof. $\Leftarrow$ direction is straightforward.

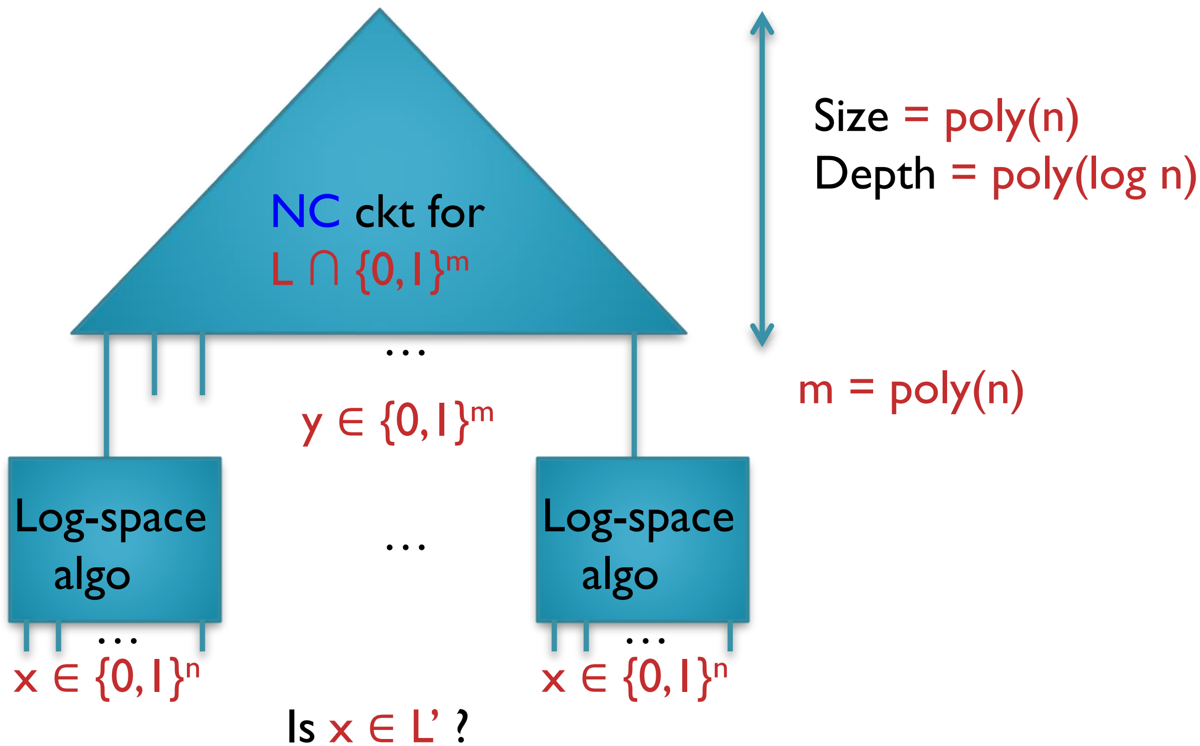
- Can't hope to get an efficient parallel algorithm for a P-complete problem unless $P \subseteq NC$.

No parallel algo for PC problems

- Theorem. Let L be a P-complete language. Then,

  L is in NC $\Leftrightarrow$ $P \subseteq NC$.

- Proof.($\Rightarrow$) Let $L' \in P$. As L is P-complete, $L' \leq_l L$.

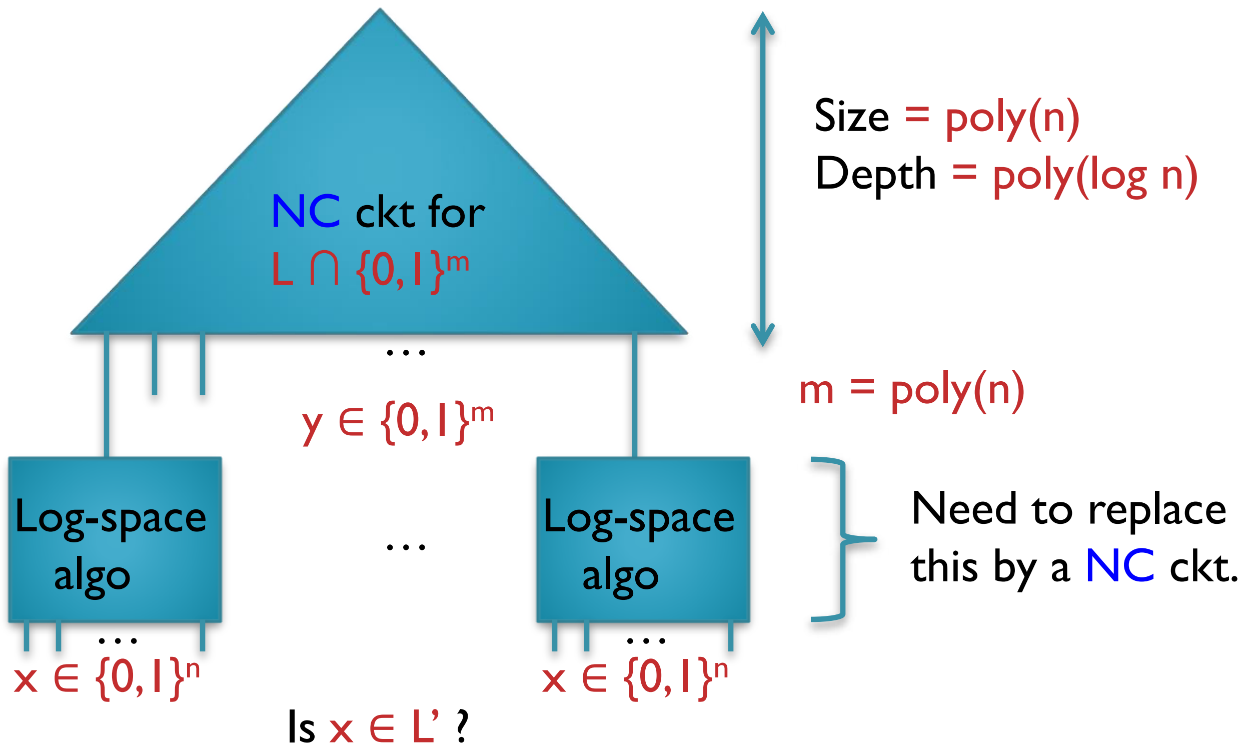


No parallel algo for PC problems

- Theorem. Let L be a P-complete language. Then,

  L is in NC $\Leftrightarrow$ $P \subseteq NC$.

- Proof.($\Rightarrow$) Let $L' \in P$. As L is P-complete, $L' \leq_l L$.



Parallelization of Log-space

- Do problems in L have efficient parallel algorithms? Yes!

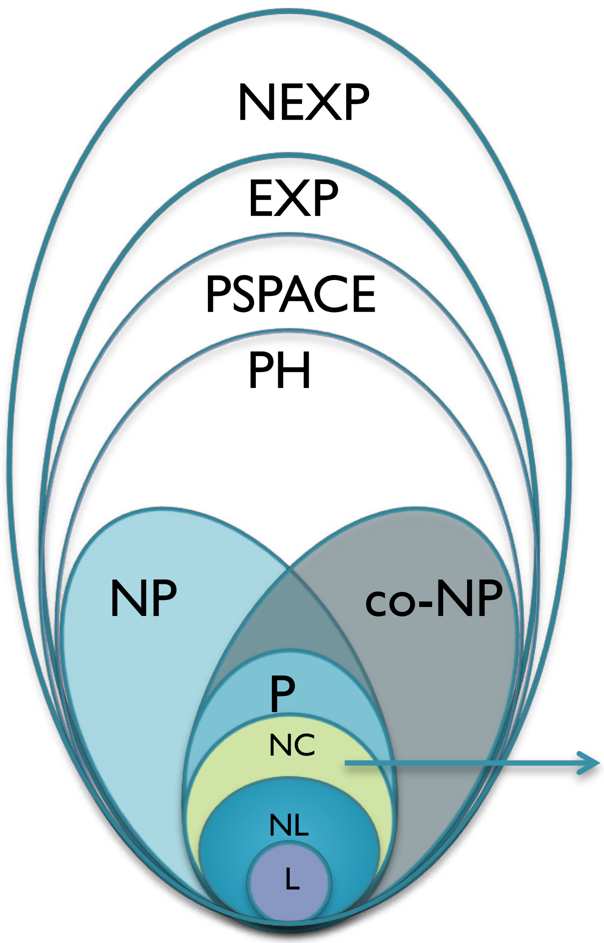
- Theorem. NL $\subseteq$ (uniform) NC. *(Assignment problem)*

Parallelization of Log-space

- Do problems in L have efficient parallel algorithms? Yes!

- Theorem. NL $\subseteq$ (uniform) NC. *(Assignment problem)*
- Proof sketch.
- 1. Construct the adjacency matrix A of the configuration graph.
- 2. Use repeated squaring of A to find out if there's a path from start to accept configurations.

Complexity zoo



(uniform) NC

In fact, (uniform) $NC^1 \subseteq L$ and $NL \subseteq$ (uniform) NC^2.
(Assignment)